Datapaths and Pipelines

CICS 515: Part 1
Summer 2003

The Processor: Datapath & Control

- We’re ready to look at an implementation of the MIPS
- Simplified to contain only:
  - memory-reference instructions: lw, sw
  - arithmetic-logical instructions: add, sub, and, or, slt
  - control flow instructions: beq, j
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
More Implementation Details

Abstract / Simplified View:

- Two types of functional units:
  - elements that operate on data values (combinational)
  - elements that contain state (sequential)

Register File

- Register 0
- Register 1
- Register n - 1
- Register n

Read register number 1
Read register number 2
Read data 1
Read data 2
Register File

Clock not shown. Each register is clocked by the same clock, as in lab 2

Building the Datapath

- Use multiplexors to stitch them together
Don’t worry about the details, I just wanted to show you that there is some control logic along with the datapath (as in Lab 3)

What is Pipelining?

• Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold

• Washer takes 30 minutes

• Dryer takes 30 minutes

• It takes 30 minutes to fold clothes

• It takes 30 minutes to put clothes into drawers
Sequential Laundry

- Sequential laundry takes 8 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP

- Pipelined laundry takes 3.5 hours for 4 loads!
Pipelining Lessons

- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup
- Time to “fill” pipeline and time to “drain” it reduces speedup
- Stall for Dependences

Estimate fastest clock we can use:

- assume delay of mux = 2ns, delay of ALU=2ns
- assume it takes 2ns after rising clock edge to read a register

Shortest clock cycle we can get away with is _______ ns
Multi-Cycle Datapath:

We can use a shorter cycle time by dividing the job into three steps.

Step 1 (clock cycle 1)
Step 2 (clock cycle 2)
Step 3 (clock cycle 3)

So each cycle can be done in 2 ns! But we need three of them.
So what’s the advantage???
Splitting the MIPS datapath into 5 Stages:

Five Execution Steps

- Instruction Fetch
- Instruction Decode and Register Fetch
- Execution, Memory Address Computation, or Branch Completion
- Memory Access or R-type instruction completion
- Write-back step

INSTRUCTIONS TAKE FROM 3 - 5 CYCLES!
Pipelining

Ideal speedup is number of stages in the pipeline. Do we achieve this?

Can pipelining get us into trouble?

structural hazards: attempt to use the same resource two different ways at the same time
- E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)

data hazards: attempt to use item before it is ready
- E.g., one sock of pair in dryer and one in washer; can’t fold until get sock from washer through dryer
- instruction depends on result of prior instruction still in the pipeline

control hazards: try to make decision before condition is evaluated
- E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in
- branch instructions
Data Hazards:

Consider the following instructions:

```
add $s0, $t0, $t1
add $t2, $s0, $t3
```

Do you see a problem?

```
Instruction fetch
Reg
ALU
Data access
Reg
```

Add $s0, $t0, $t1
Add $t2, $s0, $t3

One Solution: Compiler could add no-op instructions

```
add $s0, $t0, $t1
nop
nop
nop
add $t2, $s0, $t3
```

But this really slows us down.

Can re-order instructions (perhaps we can get some useful work done instead of executing no-ops).
- but this is tricky to deal with in a compiler
Data Hazard on r1

- Dependencies backwards in time are hazards

```
add r1, r2, r3
sub r4, r1, r3
and r6, r1, r7
or  r8, r1, r9
xor r10, r1, r11
```
Data Hazard Solution:

- “Forward” result from one stage to another

**Time (clock cycles)**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IF</th>
<th>ID/RF</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>add r1, r2, r3</td>
<td>Im</td>
<td>Reg</td>
<td>Dm</td>
<td>Reg</td>
<td></td>
</tr>
<tr>
<td>sub r4, r1, r3</td>
<td>Im</td>
<td>R</td>
<td>Dm</td>
<td>Reg</td>
<td></td>
</tr>
<tr>
<td>and r6, r1, r7</td>
<td>Im</td>
<td>Im</td>
<td>Dm</td>
<td>Reg</td>
<td></td>
</tr>
<tr>
<td>or r8, r1, r9</td>
<td>Im</td>
<td>Im</td>
<td>Dm</td>
<td>Reg</td>
<td></td>
</tr>
<tr>
<td>xor r10, r1, r11</td>
<td>Im</td>
<td>Dm</td>
<td>Reg</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Modified From: © D. Patterson, UCB, 1997

Forwarding (or Bypassing): What about Loads

- Dependencies backwards in time are hazards

<table>
<thead>
<tr>
<th>Instruction</th>
<th>IF</th>
<th>ID/RF</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw r1, 0(r2)</td>
<td>Im</td>
<td>Reg</td>
<td>Dm</td>
<td>Reg</td>
<td></td>
</tr>
<tr>
<td>sub r4, r1, r3</td>
<td>Im</td>
<td>R</td>
<td>Dm</td>
<td>Reg</td>
<td></td>
</tr>
</tbody>
</table>

- Can’t solve with forwarding:
  - Must delay/stall instruction dependent on loads

Modified From: © D. Patterson, UCB, 1997
Single Memory is a Structural Hazard

Control Hazards:

Another problem: What happens when we execute a branch?

- we don’t know if we will be taking the branch until the last step
- but by then, other instructions are in the pipeline!
- need to flush pipeline whenever we take a branch

Some processors have “delay slots”
- the next instruction after a branch is always execute
- rely on compiler to “fill” the slot with something useful

Most processors have h/w to predict if a branch will be taken
- after a branch, load pipeline with most-likely next instruction
- if we are wrong, we still have to flush
Control Hazard Solutions

Stall: wait until decision is clear
- It's possible to move up decision to 2nd stage by adding hardware to check registers as being read

Impact: 1 or 2 clock cycles per branch instruction
=> slow

Control Hazard Solutions

• Predict: guess one direction then back up if wrong
  - Predict not taken

• Impact: 1 clock cycles per branch instruction if right, 2 if wrong (right - 50% of time)
• More dynamic scheme: history of 1 branch (~ 90%)
Control Hazard Solutions

- Redefine branch behavior (takes place after next instruction) “delayed branch”

<table>
<thead>
<tr>
<th>Time (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
</tr>
<tr>
<td>Mem</td>
</tr>
<tr>
<td>Reg</td>
</tr>
<tr>
<td>Beq</td>
</tr>
<tr>
<td>Mem</td>
</tr>
<tr>
<td>Mem Reg</td>
</tr>
<tr>
<td>Misc</td>
</tr>
<tr>
<td>Mem Reg</td>
</tr>
<tr>
<td>Load</td>
</tr>
<tr>
<td>Mem Reg</td>
</tr>
</tbody>
</table>

Summary:

Datapaths can be pipelined, executing several instructions at once

Need to worry about control and data hazards:
- Can be taken care of by hardware or compiler or both!

Modern processors have an additional twist: there are several functional units (eg. ALUs)
  - can have several instructions in each stage at once
  - hardware or compiler decides which instruction will be assigned to which functional unit
  - in this case, we have to worry about “structural hazards”
    - don’t assign two instructions to the same ALU at a time